IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Dhruva Ranjan Chakrabarti	§	Art Unit:	2192
		§		
Serial No.:	10/698,509	§	Confirmation No.:	9606
		§		
Filed:	10/31/2003	§	Examiner:	Thuy Chan Dao
		§		
For:	RUN-TIME PERFORMANCE WITH CALL SITE INLINE SPECIALIZATION	§	Atty. Dkt. No.:	200314557-1 (HPC.0714US)

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF PURSUANT TO 37 C.F.R § 41.37

Sir:

The final rejection of claims 1-13 is hereby appealed.

I. REAL PARTY IN INTEREST

The real party in interest is the Hewlett-Packard Development Company, LP. The Hewlett-Packard Development Company, LP, a limited partnership established under the laws of the State of Texas and having a principal place of business at 11445 Compaq Center Drive West, Houston, TX 77707, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

II. RELATED APPEALS AND INTERFERENCES

None.

III. STATUS OF THE CLAIMS

Claims 1-13 have been finally rejected and are the subject of this appeal.

IV. STATUS OF AMENDMENTS

No amendment after the final rejection dated January 7, 2009 has been submitted; therefore, all amendments have been entered.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The following provides a concise explanation of the subject matter defined in each of the independent claims involved in the appeal, referring to the specification by page and line number and to the drawings by reference characters, as required by 37 C.F.R. § 41.37(c)(1)(v). Each element of the claims is identified by a corresponding reference to the specification and drawings where applicable. Note that the citation to passages in the specification and drawings for each claim element does not imply that the limitations from the specification and drawings should be read into the corresponding claim element.

Independent claim 1 recites a method of compiling a computer program with inline specialization (Spec., p. 26, ln. 3-5), the method comprising:

> given a call-graph (Fig. 6(b)), if multiple call-chains in the call-graph have a common call site, inlining the common call site in one or more of the call-chains, without inlining the common call site into all of said multiple call-chains having the common call site (Spec., p. 26, ln. 1 – p. 27, ln. 6; Figs. 6(b)-6(e)).

Independent claim 7 recites an apparatus for compiling a computer program with inline specialization (Spec., p. 26, ln. 3-5), the apparatus comprising:

memory (Spec., p. 5, ln. 9) configured to store computer-readable instructions and data;

a processor (Spec., p. 6, ln. 6) configured to access said memory and to execute said computer-readable instructions;

computer-readable instructions (Fig. 1A) stored in said memory and configured to inline a common call site in one or more call-chains in a call-graph, without inlining the common call site into all call-chains having the common call site (Spec., p. 26, ln. 1 – p. 27, ln. 6; Figs. 6(b)-6(e)).

Independent claim 13 recites:

a computer-readable storage medium storing a computer program (Fig. 1A) in executable form, the computer program being a source code compiler with cross-module optimization, the compiler including an inline specialization feature such that given a call-graph, if multiple call-chains in the call-graph have a common call site, the common call site is inlined in one or more of the call-chains, without having to inline the common call site into all of the multiple call-chains having the common call site (Spec., p. 26, ln. 1 – p. 27, ln. 6; Figs. 6(b)-6(e)).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. Claims 1-13 were rejected under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent No. 7,080,366 (Kramskoy) in view of U.S. Patent No. 6,651,243 (Berry).

VII. ARGUMENT

The claims do not stand or fall together. Instead, Appellant presents separate arguments for various independent and dependent claims. Each of these arguments is separately argued below and presented with separate headings and sub-headings as required by 37 C.F.R. § 41.37(c)(1)(vii).

A. Claims 1-13 were rejected under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent No. 7,080,366 (Kramskoy) in view of U.S. Patent No. 6,651,243 (Berry).

1. Claims 1, 7, 13.

The obviousness rejection of independent claim 1 is in error. To make a determination under 35 U.S.C. § 103, several basic factual inquiries must be performed, including determining the scope and content of the prior art, and ascertaining the differences between the prior art and the claims at issue. *Graham v. John Deere Co.*, 383 U.S. 1, 17, 148 U.S.P.Q. 459 (1965). Moreover, as held by the U.S. Supreme Court, it is important to identify a reason that would have prompted a person of ordinary skill in the art to combine reference teachings in the manner that the claimed invention does. *KSR International Co. v. Teleflex, Inc.*, 127 S. Ct. 1727, 1741, 82 U.S.P.Q.2d 1385 (2007).

Here, even if Kramskoy and Berry could be hypothetically combined, the hypothetical combination of the references would not have disclosed or hinted at all elements of claim 1.

Claim 1 recites a method of compiling a computer program with inline specialization, comprising:

given a call-graph, if multiple call-chains in the call-graph have a common call site, inlining the common call site in one or more of the call-chains, without inlining the common call site into all of said multiple call-chains having the common call site.

An example embodiment of the subject matter of claim 1 is illustrated in Fig. 6 of the specification, which is reproduced below:

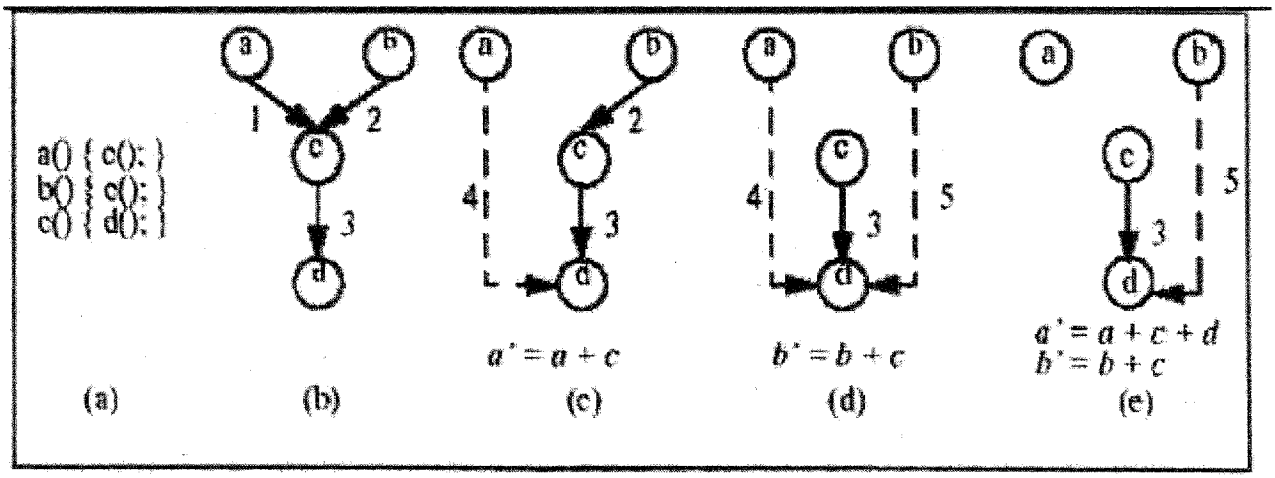


Figure 6

The text accompanying Fig. 6 is reproduced below:

We now discuss the creation of new edges and the consequences thereof in further detail. To visualize the call-relationship after a routine is inlined into another, we refer to Figure 6. Figure 6(a) shows a simple code segment. Figure 6(b) shows the corresponding call-graph. Figure 6(c) shows the call graph after a decision is taken to inline the call site I. A new edge 4, shown as a dotted edge, is added from node a (now a') to node d. This is because when node a inlines node c, the call to node d gets imported into a. This new call site, 4, is inserted into the work-list and is considered by subsequent inline analysis. Figure 6(d) shows the call-graph after a decision is taken to inline the call site 2. A new edge 5, shown as a dotted edge, is added from node b (now b') to node d. It is to be noted that subsequent to the scenario shown in Figure 6(d), the inline analysis phase could decide to inline only one of call sites 4 and 5. If call site 4 is inlined without inlining call site 5, this would achieve inline specialization, as shown in Figure 6(3). This is because node a (now a") will have inlined both nodes c and d, while node b (denoted b') will have inlined node c alone.

According to claim 1, for a given call-graph, if multiple call-chains in the call-graph have a common call site, then the common call site is inlined in one or more of the call chains, without inlining the common call site into all of the multiple call-chains having the common call site.

As purportedly disclosing the "inlining" feature of claim 1, the Office Action cited the following passages of Kramskoy: Fig. 1E; column 20, line 27 – column 21, line 20; column 39, lines 14-23; column 13, lines 44-49; column 39, lines 10-38. As explained by Kramskoy, Fig. 1E shows paths of execution through code of a method that is generally represented as 1066. Kramskoy, 20:10-11. In Fig. 1E of Kramskoy, each of the rectangles (blocks) represents a code fragment (*i.e.*, a fragment of the code of the method). *Id.*, 20:12-14. As explained elsewhere in Kramskoy, the goal of Kramskoy is to reduce the amount of code that has to be compiled in view of constraints on the amount of memory that is available to a compiler. *Id.*, 4:9-17. Thus, in Kramskoy, the compiler compiles fragments of code for the dominant path rather than the entirety of the code. *Id.*, 4:16-17, 56-60. As explained by Kramskoy, by compiling only the dominant path, "the storage overhead of storing compiled code which is rarely executed can be minimised." *Id.*, 5:12-15. The text in column 20 of Kramskoy that was cited by the Office Action accompanies Fig. 1E of Kramskoy, and describes how the compiler of Kramskoy identifies the dominant path, which is determined by following the most popular successors one block at a time. *Id.*, 13:44-49.

However, compiling less than all the code fragments of a method, as disclosed by Kramskoy, is completely different from inlining a call site (which in claim 1 is a "common call site") into one or more call-chains of a call-graph. As understood by persons of ordinary skill in the art and explained in the Background section of the present application, inlining "replaces a

call site with the called routine's code." Specification, page 1, line 34 – page 2, line 1. As further explained by the present specification, "in-line substitution eliminates call overhead and tailors the call to the particular set of arguments passed at a given call site." *Id.*, page 2, lines 1-3. The compilation of code fragments performed in Kramskoy, and in particular, the compilation of code fragments of a dominant path performed by Kramskoy, does not constitute inlining a common call site of a call-graph in one or more call-chains of the call-graph, as recited in claim 1.

In fact, Kramskoy itself provides objective evidence that the discussion accompanying Fig. 1E of Kramskoy, in which code fragments of a dominant path are compiled, is not the same as inlining, as recited in claim 1. Appendix 7 of Kramskoy in column 85 provides an example of inlining in the context of Fig. A7-3. *See* Kramskoy, 85:46-56. It is apparent that the discussion of compiling code fragments in connection with Fig. 1E of Kramskoy is different from the inlining discussion in column 85 of Kramskoy.

The secondary reference, Berry, cited by the Examiner also provides no teaching or hint of the inlining that is performed according to claim 1, where a common call site in a call-graph is inlined in one or more call-chains without inlining the common call site into all of the multiple call-chains having the common call site. Berry shows a call stack tree. Assuming for the sake of argument that a call stack tree can constitute a call-graph (which it does not), it is respectfully submitted that there is absolutely no hint given in Berry of inlining a common call site in such a call stack tree in one or more call-chains, without inlining the common call site into all of the multiple call-chains having the common call site.

Moreover, it is respectfully submitted that neither Kramskoy nor Berry discloses a call-graph as recited in claim 1. As understood by persons of ordinary skill in the art, a call-graph is

a graph that indicates relationships between routines of a program. Examples of call-graphs are shown, for example, in Fig. 2B and Fig. 6 of the present specification. As explained on page 7, lines 22-25, of the present specification: "the nodes of the call graph correspond to the program routines, and the edges of the call graph correspond to the call sites. The call site labeled '1' corresponds to the call from main() to foo(). The call site labeled '2' corresponds to the call from main() to bar() . . .."

In contrast, the call stacks disclosed by Kramskoy and Berry are data structures that store information about actively executing routines of a program. The Office Action cited Fig. A5 of Kramskoy as purportedly disclosing a call-graph. However, as specifically stated by Kramskoy, Fig. A5 depicts a call stack. Similarly, the call stack tree of Berry "reflects the call stacks recorded to date." Berry, 18:24.

The Examiner also incorrectly cited code fragments 1072, 1080, and 1084 of Kramskoy as constituting "common call sites." 1/7/2009 Office Action at 6. The blocks 1072, 1080, and 1084 in Fig. 1E of Kramskoy represent code fragments, and thus cannot properly be considered call sites that are part of call-chains in a call-graph, as recited in claim 1.

In view of the foregoing, it is clear that the hypothetical combination of Kramskoy and Berry would not have led to the subject matter of claim 1. Therefore, the obviousness rejection of claim 1 and its dependent claims is clearly erroneous.

The obviousness rejection of independent claims 7 and 13 and their respective dependent claims is also defective.

Reversal of the final rejection of the above claims is respectfully requested.

2. Claims 2, 3, 8, 9.

Claims 2 and 8 depend respectively from claims 1 and 7, and are allowable for at least the same reasons as corresponding independent claims. Moreover, claim 2 further recites that whenever a call site from routine x to routine y is inlined, **new call sites are added from routine x to all routines inlinable within routine y**. As purportedly disclosing this feature of claim 2, the Examiner cited the following passages of Kramskoy: column 79, lines 6-36; column 85, lines 48-67. The cited column 79 passage of Kramskoy refers to inlining a single implementation of an invoked method. The cited column 85 passage refers to a code section 4034 that is a compiled version of code including a call to bar 4038, where bar refers to a method that has been inlined in the code section 4034 so that bar is now contained in the code section 4034 as section 4038. However, there is absolutely no hint given in either of the cited passages of Kramskoy of adding new call sites from routine x to all routines inlinable within routine y.

The concept of adding call sites is simply missing in Kramskoy, as well as in Berry.

Therefore, claim 2 and its dependent claims are further allowable over Kramskoy and Berry for the foregoing reason. Claim 8 and its dependent claims are also similarly further allowable.

Reversal of the final rejection of the above claims is respectfully requested.

3. Claims 4-6, 10-12.

Claims 4 and 10 depend from respective base claims 2 and 8, and are therefore allowable for at least the same reasons as the corresponding claims. Claim 4 further recites adding the new call sites to a global work-list so that the new call sites are considered for inlining. The Examiner cited the following passages of Kramskoy as purportedly disclosing this feature of claim 4: column 85, lines 48-67; column 86, lines 36-61. The cited column 85 passage of

Kramskoy refers to inlining a method into a code section 4034 – however, there is absolutely no indication or hint given here of adding new call sites from a routine x to all routines inlinable within routine y, or adding such new call sites to a global work-list so that the new call sites are considered for inlining.

The column 86 passage of Kramskoy cited by the Examiner refers to loading new classes and determining if a method of the class has been overwritten. This passage provides no hint of adding new call sites (that were added whenever a call site from routine x to routine y is inlined) to a global work-list so that the new call sites are considered for inlining.

This is a further reason that claim 4 and its dependent claims are allowable over Kramskoy and Berry.

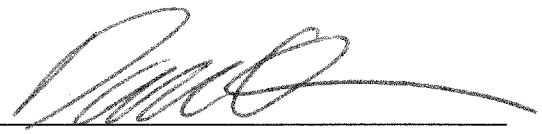
Dependent claim 10 and its dependent claims are also similarly further allowable over Kramskoy and Berry.

Reversal of the final rejection of the above claims is respectfully requested.

CONCLUSION

In view of the foregoing, reversal of all final rejections and allowance of all pending claims is respectfully requested.

Respectfully submitted,



Date: June 8, 2009

Dan C. Hu
Registration No. 40,025
TROP, PRUNER & HU, P.C.
1616 South Voss Road, Suite 750
Houston, TX 77057-2631
Telephone: (713) 468-8880
Facsimile: (713) 468-8883

VIII. APPENDIX OF APPEALED CLAIMS

The claims on appeal are:

1. A method of compiling a computer program with inline specialization, the method comprising:

given a call-graph, if multiple call-chains in the call-graph have a common call site, inlining the common call site in one or more of the call-chains, without inlining the common call site into all of said multiple call-chains having the common call site.

2. The method of claim 1, further comprising:

whenever a call site from routine x to routine y is inlined, new call sites are added from routine x to all routines inlinable within routine y.

3. The method of claim 2, further comprising:

materialization of summary information for the new call sites added to the call-graph.

4. The method of claim 3, further comprising:

addition of the new call sites to a global work-list so that the new call sites are considered for inlining.

5. The method claim 4, further comprising:

addition of dependence relationships between call sites, wherein if a new call site, y, is added because of inlining of call site, x, then y is dependent on x.

6. The method of claim 5, further comprising:

patching of the new call site, y, during inline transformation of call site, x, and

generating an intermediate transformation for the new call site, y.

7. An apparatus for compiling a computer program with inline specialization, the apparatus comprising:

memory configured to store computer-readable instructions and data;

a processor configured to access said memory and to execute said computer-readable instructions;

computer-readable instructions stored in said memory and configured to inline a common call site in one or more call-chains in a call-graph, without inlining the common call site into all call-chains having the common call site.

8. The apparatus of claim 7, wherein whenever a call site from routine x to routine y is inlined, new call sites are added from routine x to all routines inlinable within routine y.

9. The apparatus of claim 8, wherein materialization of summary information for the new call sites added to the call-graph is performed.

10. The apparatus of claim 9, wherein the new call sites are added to a global work-list so that the new call sites are considered for inlining.

11. The apparatus of claim 10, wherein dependence relationships are created between call sites.

12. The apparatus of claim 11, wherein the inline transformation patches up an intermediate representation of the new call sites (by considering the dependence relationships) before potentially inlining the new call sites.

13. A computer-readable storage medium storing a computer program in executable form, the computer program being a source code compiler with cross-module optimization, the compiler including an inline specialization feature such that given a call-graph, if multiple call-chains in the call-graph have a common call site, the common call site is inlined in one or more of the call-chains, without having to inline the common call site into all of the multiple call-chains having the common call site.

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.